PySStuBs: Characterizing Single-Statement Bugs in Popular Open-Source Python Projects

Arthur V. Kamienski, Luisa Palechor, Cor-Paul Bezemer, Abram Hindle
University of Alberta
Edmonton, Alberta, Canada
{kamiensk,palechor,bezemer,hindle1}@ualberta.ca

***Abstract*—Single-statement bugs (SStuBs) can have a severe impact on developer productivity. Despite usually being simple and not offering much of a challenge to fix, these bugs may still disturb a developer's workflow and waste precious development time. However, few studies have paid attention to these simple bugs, focusing instead on bugs of any size and complexity. In this study, we explore the occurrence of SStuBs in some of the most popular open-source Python projects on GitHub, while also characterizing their patterns and distribution. We further compare these bugs to SStuBs found in a previous study on Java Maven projects. We find that these Python projects have different SStuB patterns than the ones in Java Maven projects and identify 7 new SStuB patterns. Our results may help uncover the importance of understanding these bugs for the Python programming language, and how developers can handle them more effectively.**

***Index Terms*—Single-statement bugs, Python, open-source projects**

I. Introduction

All software developers have to deal with bugs at some point in their careers, either while working on toy projects for leisure or developing enterprise-grade software for industry. These bugs may occur due to countless reasons, from syntax errors to programming logic-related issues [11]. Bugs may also vary in size and complexity, ranging from a single wrong token to many lines of code spread across different components.

Tricky bugs that span multiple functions and statements may offer developers a great challenge to unravel, making them waste precious hours of their work time [15]. Single-statement bugs (also known as simple stupid bugs, or SStuBs) [9] also jeopardize developer productivity by interrupting their workflow, despite being easier to fix. Frequently occurring SStuBs may significantly impact workflows by continuously making developers switch contexts to fix problems.

Several studies have analyzed the impact of bugs on developer productivity and projects' lifecycles [7], [18]. Many of those have focused on automatically identifying bugs to relieve the developers' burden of manually searching and fixing them [21]. However, researchers have not given a lot of attention to SStuBs and their relevance to software development. Studying and characterizing those SStuBs can help developers in identifying them sooner, reducing the amount of time they invest in solving the problem.

Recently, a study by Karampatsis and Sutton [9] identified and analyzed SStuBs in 1,000 open-source Java Maven projects. The authors characterize 16 SStuB patterns and discuss their frequency in those projects. While that study sheds light on how SStuBs occur and the importance of studying them, it only focuses on Java Maven projects.

With that in mind, this paper identifies and analyzes the occurrence of SStuBs in a different programming language, namely 1,000 of the most popular Python projects on GitHub. The Python programming language shows several differences from other languages such as Java [6], [16], which may be reflected in the types and number of occurrences of SStuBs.

By collecting data from these Python projects and using a similar approach as the one used by Karampatsis and Sutton [9], we seek to understand the differences in SStuBs between the Java and Python projects. More specifically, we answer the following Research Questions (RQs):

RQ1. What are the most common single-statement bugs in the most popular open-source Python projects?

The differences between Python and other programming languages may result in the occurrence of different types of single-statement bugs in Python projects. In this research question, we discuss the types of SStuBs we identified in the studied Python projects. We identify the 16 top occurring patterns, and characterize 7 new patterns not found within the patterns presented by Karampatsis and Sutton [9].

RQ2. How do the single-statement bugs we identified compare to the ones found in Java Maven projects?

While Python projects might contain new SStuB patterns, we still expect to find some of the patterns described for Java Maven projects due to the similarities between the syntaxes of the two languages (e.g., control structures and arithmetic operators). In this research question, we compare the types of Python SStuBs to the ones in Java Maven projects as described by Karampatsis and Sutton [9]. We find that some of the SStuBs are unique to each programming language, which affects their frequency in the projects.

II. Methodology

In this section, we describe the methodology we used to select the studied projects, gather their data, and identify SStuBs. Figure 1 shows an overview of the steps we took.

A. Selecting Python projects

We selected the 1,000 most popular Python projects on GitHub as measured by their number of stars in January, 2021. We used GitHub's search engine to obtain a list of Python

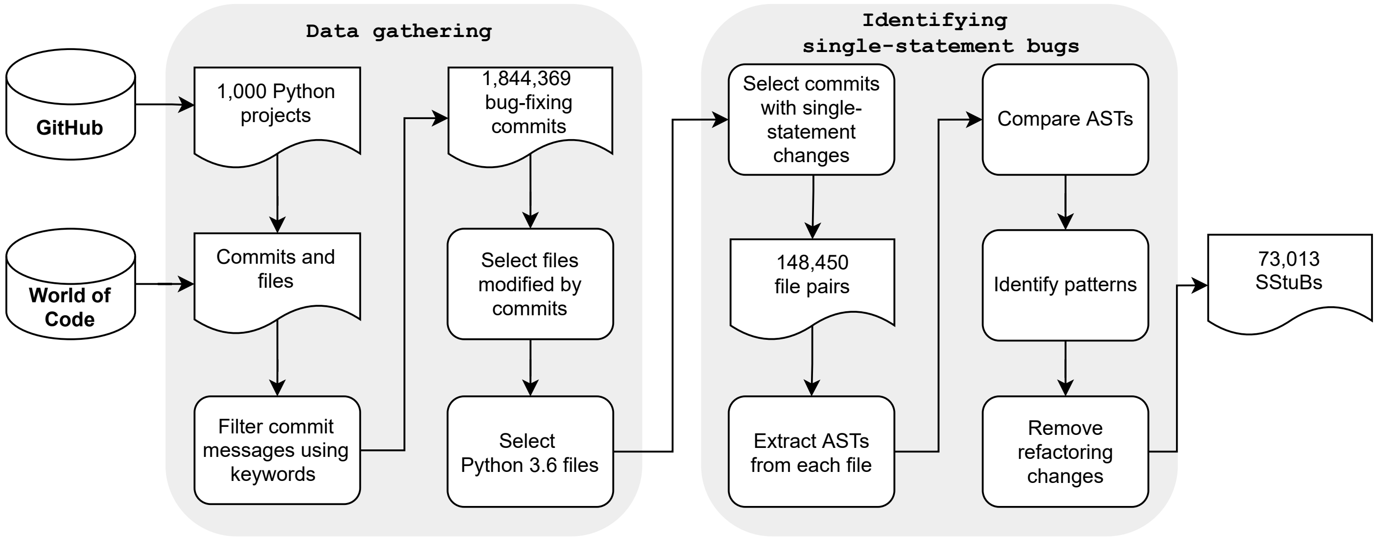
Data gathering
Identifying single-statement bugs
GitHub
World of Code
1,000 Python projects
Commits and files
Filter commit messages using keywords
1,844,369 bug-fixing commits
Select files modified by commits
Select Python 3.6 files
Select commits with single-statement changes
148,450 file pairs
Extract ASTs from each file
Compare ASTs
Identify patterns
Remove refactoring changes
73,013 SStuBs

Fig. 1. Overview of the steps taken in our methodology.

projects (i.e., projects which have most of their content written in Python) ordered by their number of stars. We chose this specific number of projects to provide a fairer comparison with the 1,000 Java Maven projects collected by Karampatsis and Sutton [9] for their “SStuBs L” dataset.

B. Gathering data

We gathered data from the projects we selected in Section II-A using World of Code (WoC) [14], an infrastructure for mining open-source software and their version control data which is updated on a monthly basis. Starting from the project IDs on GitHub, we used the WoC API to collect the commits associated to them. We used a similar process to gather the files containing source code associated with each commit. The data was collected in January, 2021.

We note that the data we collected from WoC is not an exact representation of the projects, as the API could not retrieve some of the entities belonging to the projects. For example, we could not collect some of the commits referenced by the projects. However, we measured an overall loss of data of less than 10%, which should not affect our overall results.

We collected 6,062,534 commits from the 1,000 projects we selected. We identified bug-fixing commits using the same methodology described by Karampatsis and Sutton [9], i.e., by filtering commit messages which contain one of the following keywords: ‘error’, ‘bug’, ‘fix’, ‘issue’, ‘mistake’, ‘incorrect’, ‘fault’, ‘defect’, ‘flaw’, and ‘type’. Using this method, we obtained 1,844,369 bug-fixing commits. In addition, we excluded commit messages that included the ‘refactor’ keyword to help reduce the number of false positives in our sample, but those only accounted for 1% (18,156) of the commits. Lastly, we filtered out any commits that added or deleted files.

We moved on to gather the files referenced by each commit. In this step, we collected the files before and after they were modified by the commit. As the projects can also contain files that are not written in Python, we filter out any files which do not contain the suffix ‘.py’ in their file name. We also excluded any files not containing a valid Python 3.6 syntax, as those could not be parsed into Abstract Syntax Trees (ASTs) in future steps using our version of Python 3.

C. Identifying single-statement bugs

We followed a similar approach as Karampatsis and Sutton [9] to identify SStuBs. First, we used the Unix ‘diff’ command to identify the line changes between each file pair. We selected only the pairs which showed single-statement changes, while discarding those containing line deletions and additions. We also discarded all of the files from commits that showed multiple-statement changes in any position in any file. After this step, we were left with 148,450 file pairs.

For file pairs that contain multiple single-statement changes, we derived new pairs by applying each of the changes to the original file, one at a time. Thus, each new pair contained only one change. We parsed the files of each pair using Python’s *ast* library, yielding an Abstract Syntax Tree (AST) for each of them. The *ast* library ignores comments and whitespaces, and we therefore do not consider changes to those in our analysis. We also ignore changes to class and function docstrings.

We wrote a custom Python script to compare the resulting AST pairs. Using the script, we perform a simultaneous depth-first traversal of each tree and locate the first pair of nodes in which the trees differ. Each pair of diverging nodes thus corresponds to a single-statement change in between the files. We manually analyzed each type of node differences to identify if they matched any of the SStuB patterns as described by Karampatsis and Sutton [9]. However, unlike Karampatsis and Sutton [9], our method only matches the first pattern found for a statement and not all of them. Furthermore, we analyzed the most common types of node differences to identify the 16 most common SStuBs, and found 7 new SStuB patterns (described in Section III-A).

D. Removing refactoring changes

We noticed that some of the patterns we identified in Section II-C described changes to function and class definitions, such as their names and arguments. As those patterns likely relate to the refactoring of code and not to bug-fixing changes, we decided to exclude them from our analysis. We also excluded changes made to statements that referenced those refactored entities in any of the files that originated from the same commit. For example, if a commit changed the name of a class in one of its files, we excluded all of the changes made to that class' usages across all of the files in the commit.

We also observed that many of the single-statement changes we identified describe changes to the values of string constants. Strings in Python serve a large number of purposes, from indexing values in dictionaries to storing data, and developers may need to frequently change them to account for new code versions. Strings also have a flexible length and can contain any type of written text, and are therefore prone to errors and misspellings. However, not all of the changes to those string values can be considered bugs, and including them in our analysis may introduce many false positives. For example, developers frequently use hard-coded strings in natural language to write messages that describe errors, program functionalities or interactions with users, and changes to those strings may not change the behaviour of the program. We therefore excluded these changes from our analysis.

We note that we only identified trivial changes, and we did not remove more complex refactorings. In the end, we were left with 126,912 single-statement changes that altered the ASTs, 58% (73,013) of which belonging to the 23 SStuB patterns we used. The remaining changes did not fit any of our patterns. Our final dataset with 73,013 SStuBs is publicly available online [8].

III. Results

In this section we answer our two RQs by describing the results of our analysis of the 73,013 SStuBs and 23 patterns we obtained from Section II.

A. RQ1. What are the most common single-statement bugs in the most popular open-source Python projects?

We found 7 new SStuB patterns in Python projects. Out of the 23 SStuBs we identified, 7 were not previously defined by Karampatsis and Sutton [9]. While some of these patterns may occur in Java, others only occur due to the difference in syntax between the languages. We give a brief description of each of these new patterns below. The number of occurrences of each pattern can be seen in Table I.

- *Change Attribute Used* - When developers change the attribute accessed from an object. For example, `person.name` changes to `person.age`.
- *Add Function Around Expression* - When developers put an expression inside a function call, often for modifying the returned value. For example, `human = person` changes to `human = is_human(person)`.
- *Add Elements to Iterable* - When developers add an element to a hard-coded iterable, such as a `list` or a `tuple`. For example, `info = (name, age)` changes to `info = (name, age, height)`.
- *Change Keyword Argument Used* - When developers change the keyword argument used in a function call or object instantiation. For example, `Person(name=20)` changes to `Person(age=20)`.
- *Add Method Call* - When developers add a method call to an expression which references an object, changing the return value. For example, `year = person` changes to `year = person.birth_year()`.
- *Change Constant Type* - When developers change the type of a hard-coded constant. For example, `person.age = '10'` changes to `person.age = 10`.
- *Add Attribute Access* - When developers access the attribute of an object instead of the object itself. For example, `say_hello_to(person)` changes to `say_hello_to(person.name)`.

TABLE I
Counts of SStuB patterns in Python and Java Maven projects. Patterns in bold indicate the new patterns we identified. Numbers in bold show the patterns that occur over two times more in Java.

Pattern name	Python	%	Java [9]	%
Same Function More Args	9,958	14	5,100	8
Wrong Function/Method Name	9,091	12	10,179	16
Change Identifier Used	8,973	12	22,668	**35**
Add Function Around Expression	6,363	9	0	0
Change Attribute Used	5,229	7	0	0
Change Numeric Literal	4,775	7	5,447	8
Change Operand	4,657	6	807	1
Same Function Less Args	3,381	5	1,588	2
Add Method Call	3,338	5	0	0
Add Elements to Iterable	2,541	3	0	0
More Specific If	2,443	3	2,381	4
Change Constant Type	2,199	3	0	0
Change Unary Operator	2,187	3	1,016	2
Change Keyword Argument Used	1,554	2	0	0
Change Boolean Literal	1,466	2	1,842	3
Add Attribute Access	1,439	2	0	0
Same Function Wrong Caller	1,163	2	1,504	2
Change Binary Operator	976	1	2,241	**5**
Less Specific If	943	1	2,813	**4**
Same Function Swap Args	336	>1	612	1
Change Modifier	0	0	5,011	**8**
Delete Throws Exception	0	0	508	**1**
Missing Throws Exception	0	0	206	**>1**
Total	73,013	100	63,923	100

B. RQ2. How do the single-statement bugs we identified compare to the ones found in Java Maven projects?

The studied Python and Java Maven Projects share most of the 16 original SStuBs. We could find 13 of the 16 SStuBs identified in Java Maven projects in Python projects, although in different proportions. We applied a Chi-squared (χ^2) test to the SStuB categories found both in Java and Python and found that the difference in proportion of SStuB types was statistically significant ($p < .001$).

We observed this difference in patterns such as *Wrong Function/Method Name* (as seen in a commit from the Keras project [4] with a change from `model.train` to `model.fit`), which comprised 16% (10,179) of the bugs in Java and 12% (9,091) of the bugs in Python.

While shared patterns can occur in both languages, differences between the syntax and type system of the two programming languages make it impossible for the other three patterns to occur in Python. Therefore, we did not observe any *Change Modifier*, *Missing Throws Exception* or *Delete Throws Exception* SStuBs. The Python programming language does not have access level modifiers (e.g., *public* and *private*) and developers instead use naming conventions to simulate the access restriction behaviour. Similarly, there is no way to explicitly denote that a function *throws* an exception.

Many of the new SStuBs we identified relate to Python's dynamic type system. We observed that many of the SStuB patterns we identified in Python can be linked to the flexible way in which it allows developers to work with data types.

In Python, variables can be created and assigned a value without an explicit type declaration, and then later be reassigned a new value of a different type. This dynamic typing system allows for patterns such as *Change Constant Type*, exemplified by a commit in the Django project [3] when the value assigned to a variable was changed from a string to an integer (`param = "1"` to `param = 1`).

Other examples are *Add Method Call* and *Add Attribute Access*, which can only occur if the change from an object reference to a return value is allowed. We observed this pattern in a commit in the Ansible project [2], when `base.group_upgrade(group)` was changed to `base.group_upgrade(group.id)`. Similarly, the *Add Function Around Expression* pattern is usually related to changing the value and type of an expression. We could observe such a change in the Scipy project [1], in which a variable was cast to an integer when being returned from a function (`return nnz` changed to `return int(nnz)`).

In contrast, Java has a static type system that checks for type inconsistencies during program compilation. While this system still allows for changes in the value of variables and constants of the same type (described by patterns such as *Change Identifier Used* and *Change Numerical Literal*), it prevents the occurrence of the patterns mentioned above.

IV. Threats to validity

Internal validity. We selected the 1,000 Python projects based on their number of stars as a measure of popularity. However, there are other ways to measure the popularity of a project (e.g., the number of forks and contributors) which could lead to the selection of a different set of projects.

Construct validity. We are limited by the accuracy of the data provided by the GitHub project search. This may have excluded some relevant projects from our analysis, including projects that are more popular than the ones we selected.

Despite checking commits and files for refactoring changes, we could not detect all of them and the number of SStuBs may be overestimated. For example, we did not check for the renaming of entities such as variables. Others have shown that identifying refactorings in Python is complicated due to Python's dynamic nature [22]. As Python is a dynamically typed language, refactoring Python code tends to cause more errors than in statically typed languages like Java [17].

External validity. Our results are limited to popular open-source Python projects and may not generalize to other programming languages, or even Python code from other sources. However, many of our findings overlap with the ones from Karampatsis and Sutton [9], which may indicate general trends for other programming languages and projects.

V. Related work

Prior work has also focused on detecting bugs using AST representations. Karampatsis and Sutton [9] found single-statement bugs by mining a set of 100 and 1,000 open-source Java projects. The authors used ASTs extracted from the modified files before and after the bugs were fixed, finding that around 33% of the fixes could be described with their patterns. Martinez and Durieux [20] developed repair tools for Python, presenting an empirical study to repair the QuixBugs benchmark, even though the authors focus on Java implementations. Zhaogui and Liu [19] proposed a predictive analysis of Python projects by collecting traces and detecting bugs. They evaluated their prototype on 11 Python projects and find 46 bug types. Chen and Lin [13] used ASTs to study fine-grained source code changes in Python and, later on, analyzed the dynamic feature of Python code when fixing bugs [5]. Other studies investigated program repair patterns and some of the patterns we use in our work have been used by Le Goues et al. [12], Kim et al. [10], and Karampatsis and Sutton [9].

This research differs from those mentioned above in that we detect single-statement bugs in 1,000 Python projects and discuss them with single-statement bugs from Maven projects found by Karampatsis and Sutton [9].

VI. Conclusion

In this paper we analyze the most common single-statement bugs in Python code using data from some of the most popular open-source Python projects on GitHub. We selected projects based on their number of stars and used World of Code (WoC) to collect commit messages and files. After preprocessing the data, we compared the Abstract Syntax Trees (ASTs) for pairs of files before and after the bug fixes. As a result, we identified 23 "Simple Stupid Bug" (SStuB) patterns and 73,013 changes that matched those patterns. Additionally, we characterize 7 new SStuB patterns found in the studied Python projects. We moved on to compare the SStuBs we found to the ones found by Karampatsis and Sutton [9], showing that differences in the programming languages, and style of typing (dynamic versus static) change the types of SStuBs identified. Our findings may be used as a way of understanding these types of bugs occurring in Python code, and may help developers by improving the way they handle them. We also share our dataset online [8], allowing its use in future research.

References

[1] "Commit on GitHub: BUG: Homogenezie nnz type to be int for all sparse matrix types." [Online]. Available: https://github.com/scipy/scipy/commit/42420f14c14a622149024a6514b8b7f10620d5bc

[2] "Commit on GitHub: Fix using DNF group upgrade/remove api." [Online]. Available: https://github.com/ansible/ansible/commit/7a6c5dd1ab079d301f41c9557c50557cd0a69029

[3] "Commit on GitHub: Fixed #23434 – Coerce Oracle bool params to int." [Online]. Available: https://github.com/django/django/commit/4298f6261060a13eff1e30e6119c2ad5d64e5700

[4] "Commit on GitHub: Fixed import errors with six.moves.cPickle and model.train typo in the skipgram embeddings example." [Online]. Available: https://github.com/keras-team/keras/commit/034822359d9a4c3b2fc63de0676bad512b426112

[5] Z. Chen, W. Ma, W. Lin, L. Chen, Y. Li, and B. Xu, "A study on the changes of dynamic feature code when fixing bugs: towards the benefits and costs of python dynamic features," *Science China Information Sciences*, vol. 61, 01 2018.

[6] G. Destefanis, M. Ortu, S. Porru, S. Swift, and M. Marchesi, "A statistical comparison of java and python software metric properties," in *Proceedings of the 7th International Workshop on Emerging Trends in Software Metrics*, 2016, pp. 22–28.

[7] L. A. F. Gomes, R. da Silva Torres, and M. L. Côrtes, "Bug report severity level prediction in open source software: A survey and research opportunities," *Information and software technology*, vol. 115, pp. 58–78, 2019.

[8] A. V. Kamienski, L. Palechor, C.-P. Bezemer, and A. Hindle, "PySStuBs: Single-Statement Bugs in Popular Open- Source Python Projects," Jan. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4589607

[9] R.-M. Karampatsis and C. Sutton, "How often do single-statement bugs occur? the manysstubs4j dataset," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 573–577.

[10] D. Kim, J. Nam, J. Song, and S. Kim, "Automatic patch generation learned from human-written patches," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 802–811.

[11] A. J. Ko and B. A. Myers, "A framework and methodology for studying the causes of software errors in programming systems," *Journal of Visual Languages & Computing*, vol. 16, no. 1-2, pp. 41–84, 2005.

[12] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer, "Genprog: A generic method for automatic software repair," *Ieee transactions on software engineering*, vol. 38, no. 1, pp. 54–72, 2011.

[13] W. Lin, Z. Chen, W. Ma, L. Chen, L. Xu, and B. Xu, "An empirical study on the characteristics of python fine-grained source code change types," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 188–199.

[14] Y. Ma, C. Bogart, S. Amreen, R. Zaretzki, and A. Mockus, "World of code: an infrastructure for mining the universe of open source vcs data," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 143–154.

[15] S. Planning, "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology*, 2002.

[16] L. Prechelt, "An empirical comparison of c, c++, java, perl, python, rexx and tcl," *IEEE Computer*, vol. 33, no. 10, pp. 23–29, 2000.

[17] M. Schäfer, "Refactoring tools for dynamic languages," in *Proceedings of the Fifth Workshop on Refactoring Tools*, ser. WRT '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 59–62. [Online]. Available: https://doi-org.login.ezproxy.library.ualberta.ca/10.1145/2328876.2328885

[18] J. Uddin, R. Ghazali, M. M. Deris, R. Naseem, and H. Shah, "A survey on bug prioritization," *Artificial Intelligence Review*, vol. 47, no. 2, pp. 145–180, 2017.

[19] Z. Xu, P. Liu, X. Zhang, and B. Xu, "Python predictive analysis for bug detection," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 121–132. [Online]. Available: https://doi.org/10.1145/2950290.2950357

[20] H. Ye, M. Martinez, T. Durieux, and M. Monperrus, "A comprehensive study of automatic program repair on the quixbugs benchmark," in *2019 IEEE 1st International Workshop on Intelligent Bug Fixing (IBF)*, 2019, pp. 1–10.

[21] T. Zhang, H. Jiang, X. Luo, and A. T. Chan, "A Literature Review of Research in Bug Resolution: Tasks, Challenges and Future Directions," *The Computer Journal*, vol. 59, no. 5, pp. 741–773, 05 2016. [Online]. Available: https://doi.org/10.1093/comjnl/bxv114

[22] W. Zhou, Y. Zhao, G. Zhang, and X. Shen, "Harp: Holistic analysis for refactoring python-based analytics programs," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, 2020, pp. 506–517.